Interactive Computation and Visualization Towards a Virtual Wind Tunnel

Jérémie Labroquère, Régis Duvigneau, Thibaud Kloczko, Julien Wintz

INTERACTIVE COMPUTATION AND VISUALIZATION TOWARDS A VIRTUAL WIND TUNNEL

J. Labroquère[1], R. Duvigneau[1], T. Kloczko[2] & J. Wintz[2]

INRIA Sophia-Antipolis Méditerannée, 2004 toute des lucioles, 06902 Sophia-Antipolis, France
[1] OPALE Project-Team, Jeremie.Labroquere@inria.fr, Regis.Duvigneau@inria.fr
[2] Experimentation and Development Team, Thibaud.Kloczko@inria.fr, Julien.Wintz@inria.fr

INTRODUCTION

Computational Fluid Dynamics (CFD) software is now commonly used for aerodynamic studies as a replacement of traditional experimental facilities. CFD codes take benefit from the increase of the computational facilities and the maturity of numerical methods. However, they are usually based on old-fashioned software architectures, inherited from the 80s. They are typically built as static codes, independently from software components that are used to construct the geometry and the grid, or visualize flow fields. As a consequence, the whole simulation process is usually quite complex, including several different phases, and is restricted to expert users. Moreover, it is not possible for the user to interact with the computed flow, which makes a significant difference with an experimental wind tunnel. In the latter case, the user can for instance adjust inflow parameters, or modify some geometrical characteristics such as wing incidence, while interactively observing the flow evolution. The studies on interactive CFD simulation are quite uncommon, and a very few papers can be found on this topic [6, 7, 10].

In this study, we investigate the use of a modern software architecture in the context of CFD, which allows the user to interact with the computation, by modifying physical or numerical parameters during the computation and visualize the impact on the flow. Our objective is to evaluate the interest of such a software architecture and measure the possible benefit for scientific studies.

We describe in a first section the proposed software architecture and its new features, that allow the user to interact with the computation and visualization. Then, the application to compressible flow simulation is considered. In particular, we show that various implementations can be envisaged, with different advantages and drawbacks. The possible interactions with the computation and visualization are described. In a third section, some illustrations are proposed. Finally, the benefit of the use of these new features are discussed from the point of view of the CFD practitioner.

1. SOFTWARE ARCHITECTURE

1.1 Platform and plugins system

Traditionally, CFD codes are only composed of a main program built with some static libraries. The main program describes more or less the workflow that corresponds to the simulation, whereas the librairies contain the functions necessary for each task of the workflow. This straightforward program architecture has been set up during the 70s and 80s and has not really be questioned so far, although the complexity of the computations has grown significantly.

However, it appears clearly that this architecture has several drawbacks. First, the achievement of numerical experiments requires modularity, which allows to easily test different numerical methods (e.g. Roe flux v.s. HLLC flux), or modify physical parameters (e.g. inlet velocity profile). This ability to change some parts of the code is critical to precisely benchmark numerical methods or models. Obviously, the traditional static architecture is not well suited to this purpose: usually, the implementation of a new method *foo2* yields the addition of new functions and variables, that will juxtapose the existing ones *foo0*, *foo1*. Then, the code is always growing and becoming less and less readable.

Developments in CFD are now mainly collaborative projects, because it is more and more difficult for one isolated person to master all modeling, numerical and computational aspects related to complex simulations. Unfortunately, the traditional architecture is a real burden for collaborative development: since these codes are always growing (as explained above), the exchange of pieces of code and the maintenance is time consuming for long term and large scale projects. Typically, for each new method *foo2*, someone should verify if the proposed implementation is correct and will not generate conflicts with all other existing methods *foo0*, *foo1* (for instance in case of addition of an argument).

Finally, the complexity of these codes make the introduction of a new user tedious and time consuming: to modify a unique method, a new user should usually understand a large part of the code, even if it is not of interest for his study.

To overcome these limitations, we have initiated the NUM3SIS project (`http://num3sis.inria.fr`), whose software architecture is based on the distinction between the *platform* and *plugins*. The platform, written in C++ language, gathers all components dedicated to numerical simulation, that are considered as common to various computations. In practice, it consists of a set of abstractions, that can represent data or processes, commonly used in simulation. For instance, the core of the platform is composed of abstractions for grids, fields, flux computations, finite-elements, etc. As abstraction, the numerical methods related to these objects are not implemented in the platform.

On the contrary, a plugin contains a possible implementation of an abstraction defined in the platform. Note that a library written in a different language can be embedded into a plugin. In practice, plugins are dynamic libraries used by the platform at runtime. For instance, *foo0*, *foo1* and *foo2* can be three different plugins (possibly based on existing libraries) implementing the abstraction of the method *foo*, defined in the platform (see Fig. 1). This approach has several advantages: first, all the methods are not aggregated in a unique code, which improves readability and modularity. Then, it proposes a nicer framework for collaborative development, because development of new plugins can be conducted independently from the platform or other plugins. Moreover, templates can be proposed to speed-up the coding phase of plugins. Finally, an easy benchmarking procedure can be carried out by implementing the methods to be compared into different and independent plugins. One should underline that it is possible to change a parameter of a plugin, or the plugin itself, at runtime. This modifies the forthcoming computation and allows the user to interact with the simulation. This point will be detailed latter, with some examples.

1.2 Visual programming

As already mentioned, it is not easy for a new user to implement a method in an existing CFD code. This is particularly the case for people who are more familiar with mathematics than programming. This is dommageable because it is a real obstacle to the improvement of numerical methodologies.

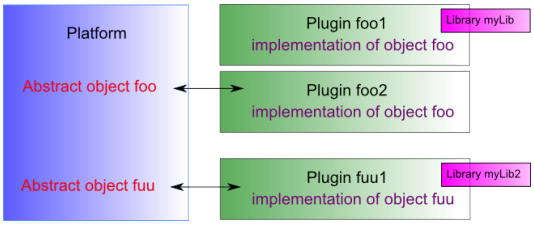

Figure 1: Illustration of plugin system.

Therefore, we have introduced a visual programming tool in the platform, that allows to build a high-level computational scenario without writing any line of code. Thus, for each abstraction defined in the platform, a visual wrapping (*node*) is introduced, which can be handled in the *composer* space, and connected to other nodes to construct the desired computational scenario (see Fig. 2). For a given node, several plugins may exist, that correspond to various implementations, and can be selected using the composer. This approach is hierarchical, in the sense that a node can contain itself a composition based on other nodes. Moreover, particular nodes represent control structures, such *for* loops, *if then* conditions, etc.

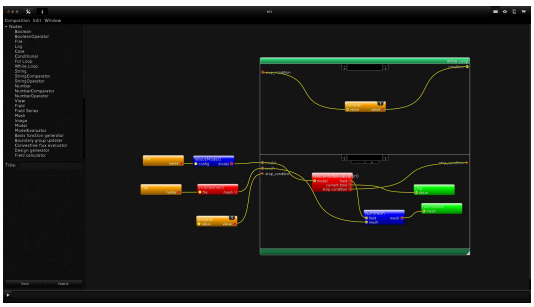

Figure 2: Illustration of the visual programming system.

This visual programming approach really facilitates the introduction of a new user. Moreover, it represents a valuable tool to prototype a new computational scenario. It should be underlined that the frontier between visual programming (using the composer) and inline programming (implementation in plugins) completely depends on the choice of the user: it is possible to program entirely the simulation process using the composer and, on the contrary, it is also possible to embed a whole simulation process into a unique plugin. Of course, an intermediate choice is usually more useful. Performance studies have been conducted to quantify a possible loss of performance due to

the use of such a visual programming tool. However, it has been found that this loss of performance is negligible.

1.3 Graphical interface and scientific visualization

The graphical interface is composed of four main spaces: the *composer space* that allows to visually build a computational scenario, the *visualization space* used for scientific visualization of computational results and two other spaces dedicated to data exchange and parallel computing.

The composer space permits the user to select a node among a set of existing nodes, drop it in a working space, select the desired implementation in a corresponding plugin (see Fig. 2), and connect it with other nodes to construct a given computational scenario.

The visualization tool is based on the well-known VTK library (`http:://www.vtk.org`) and contains all usual features to visualize scalar or vectorial fields, glyphes or streamlines, grid nodes, faces, edges, etc (see Fig. 3). The main important feature is that an object can be visualized as soon as it is computed. Actually, the view itself is considered as a node in the composition which is fed by grids, fields, etc. Therefore, each time a field is updated, the modifications can be seen in the visualization space. This feature, associated with the possibility to change a parameter or a plugin at runtime, makes the interactive computation and visualization possible.

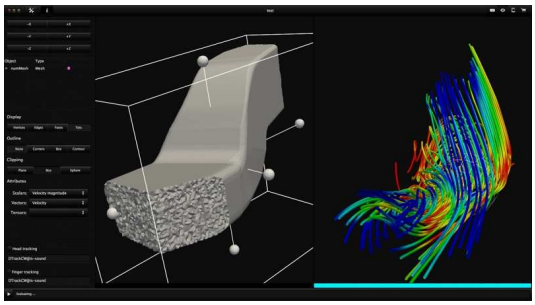

Figure 3: Illustration of the visualization space.

1.4 Stereoscopic visualization

The researches related to methods used to visualize 3D fields have known a growing interest for the last years. Indeed, simulated flows are more and more complex and the use of sophisticated turbulence models, such as LES (Large Eddy Simulation), DES (Detached Eddy Simulation) or VMS (Variational Multi-Scale) models requires visualization tools that help to capture the main characteristics of the flow. Among all the possible approaches, the use of virtual reality facilities is explored. INRIA Sophia Antipolis Méditerranée center has an *immersive space* facility, which regroups two display devices, a *CadWall* for its ease of use and an *iCube* for its immersion quality. This facility is dedicated to research in virtual reality, but can also be used by non-specialists to explore the possibilities of these new visualization devices.

The CadWall employed for the present experiments can be seen as a single screen of 3528x1200 pixels. Two images are simultaneously projected onto the wall and allows to create a 3D perception of the objects by using specific glasses. A dedicated software layer has been introduced in the platform in order to make the generation of such a stereoscopic view possible. Then, the platform can be used with the CadWall, or with any classical screen, without any modification.

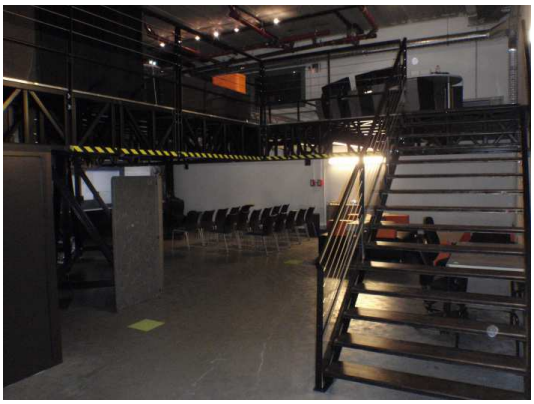

Figure 4: Virtual reality facility at INRIA Sophia Antipolis Méditerranée center.

2. APPLICATION TO COMPRESSIBLE FLOW SIMULATION

We explain in this section how the proposed software architecture is used for compressible flow simulation. We present first the numerical methods employed and then how they are implemented in the platform and plugins. Note that the platform is not devoted to CFD, fluid mechanics is only an application among others. At the present time, the platform is used by two INRIA Project-Teams for computations in aerodynamics (finite-volume method), electromagnetics (discontinuous Galerkin method), pedestrian traffic modeling (finite-volume method) and thermal conduction (finite-element method). Moreover, the computational scenario is not restricted to simulation, but other computations can be carried out, on the basis of the same software components.

2.1 Modelling and numerics

Modeling The equation solved for compressible flows simulation are the Navier-Stokes equations. In conservative form, they can be written as:

$$\partial_t \boldsymbol{W^c} + \boldsymbol{\nabla} \cdot \boldsymbol{\mathcal{F}}(\boldsymbol{W^c}) = \boldsymbol{\nabla} \cdot \boldsymbol{\mathcal{N}}(\boldsymbol{W^c}) + \boldsymbol{\mathcal{S}}(\boldsymbol{W^c}) \quad (1)$$

with the conservative variables $\boldsymbol{W^c} = (\rho, \rho u, \rho v, \rho w, \rho E)^T$, the inviscid flux $\boldsymbol{\mathcal{F}}(\boldsymbol{W^c}) = (\boldsymbol{F}(\boldsymbol{W^c}), \boldsymbol{G}(\boldsymbol{W^c}), \boldsymbol{H}(\boldsymbol{W^c}))^T$, the viscous flux $\boldsymbol{\mathcal{N}}(\boldsymbol{W^c}) = (\boldsymbol{R}(\boldsymbol{W^c}), \boldsymbol{S}(\boldsymbol{W^c}), \boldsymbol{T}(\boldsymbol{W^c}))^T$ and the source term $\boldsymbol{\mathcal{S}}(\boldsymbol{W^c})$.

The components of the inviscid flux in the global frame $\mathcal{R}_0(\widehat{\boldsymbol{x}}, \widehat{\boldsymbol{y}}, \widehat{\boldsymbol{z}})$ are:

$$\boldsymbol{F}(\boldsymbol{W^c}) = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ u(\rho E + p) \end{pmatrix}, \boldsymbol{G}(\boldsymbol{W^c}) = \begin{pmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ \rho vw \\ v(\rho E + p) \end{pmatrix}$$

$$\boldsymbol{H}(\boldsymbol{W^c}) = \begin{pmatrix} \rho w \\ \rho wu \\ \rho wv \\ \rho w^2 + p \\ w(\rho E + p) \end{pmatrix}$$

The components of the viscous flux are:

$$\boldsymbol{R}(\boldsymbol{W^c}) = \begin{pmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xz} \\ u\tau_{xx} + v\tau_{xy} + w\tau_{xz} + q_x \end{pmatrix}$$

$$\boldsymbol{S}(\boldsymbol{W^c}) = \begin{pmatrix} 0 \\ \tau_{yx} \\ \tau_{yy} \\ \tau_{yz} \\ u\tau_{yx} + v\tau_{yy} + w\tau_{yz} + q_y \end{pmatrix}$$

$$\boldsymbol{T}(\boldsymbol{W^c}) = \begin{pmatrix} 0 \\ \tau_{zx} \\ \tau_{zy} \\ \tau_{zz} \\ u\tau_{zx} + v\tau_{zy} + w\tau_{zz} + q_z \end{pmatrix}$$

To close the equations, the pressure is modeled by the perfect gas state law $E = \frac{p}{(\gamma-1)\rho} + \frac{1}{2}\boldsymbol{V} \cdot \boldsymbol{V}$, the heat flux $\boldsymbol{q}$ is modeled by using a Fick or Fourier law $\boldsymbol{q}(\epsilon, \kappa) = -\frac{\gamma\mu}{\mathrm{Pr}}\boldsymbol{\nabla}\epsilon$, the adiabatic index is set to $\gamma = 1.4$, the Prandtl number is set to $\mathrm{Pr} = 0.72$ and the viscosity μ is either assumed to be constant or modeled with the Sutherland law. By neglecting the viscous fluxes and the source term in the equation 1, the Euler equations are recovered.

Spatial discretization A Mixed finite-Element/finite-Volume (MEV) discretization is used, which consists in discretizing the domain with a mixed finite-element/finite-volume approach of vertex centered type. The inviscid fluxes are discretized with a finite-volume approach while the viscous fluxes are discretized with a finite-element approach [2, 3].

A polygonal bounded domain $\Omega \subset \mathbb{R}^n$ is considered with a bound Γ, sub-divided into a tetrahedrization or triangulation $\mathcal{T}_h$ with elements T_i. Around each vertex s_i a finite-volume control cell C_i of a measure $m(C_i)$ is constructed. The set of vertices which are joined to the vertex s_i is denoted by $\mathcal{N}(s_i)$. The subset of all the highest topological dimension polygons sharing the vertex s_i is denoted by $\mathcal{T}(s_i)$.

The inviscid fluxes are computed on the dual control cells C_i while the viscous fluxes are computed on the elements T_i. A weak formulation of the Navier-Stokes equations can be expressed using a Galerkin approach.
By integrating the equation 1 over a control cell S_i (dual control cell or element) against a regular test function φ_i, the weak formulation is written as:

$$\int_{S_i} (\partial_t \boldsymbol{W^c} + \boldsymbol{\nabla} \cdot \boldsymbol{\mathcal{F}}(\boldsymbol{W^c}))\varphi_i d\Omega = \int_{S_i} (\boldsymbol{\nabla} \cdot \boldsymbol{\mathcal{N}}(\boldsymbol{W^c}))\varphi_i d\Omega + \int_{S_i} (\boldsymbol{\mathcal{S}}(\boldsymbol{W^c}))\varphi_i d\Omega \quad (2)$$

The finite-volume method can be interpreted as a Galerking method with the control cell $S_i = C_i$ and with the test function equals 1 inside the dual control cell and 0 outside. This test function, related to the control cell C_i is defined as:

$$\varphi_i^C(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \boldsymbol{x} \text{ is in } C_i \\ 0, & \text{else} \end{cases}$$

The variables $\boldsymbol{W^c}$ are considered to be constant on each control cells C_i. These constants are denoted by $\boldsymbol{W_i^c}$ on the cell C_i (see Fig. 5 and Fig. 6).

By using the Green-Ostrogradski theorem, the left term of 2 becomes:

$$\int_{C_i} (\partial_t \boldsymbol{W^c} + \boldsymbol{\nabla} \cdot \boldsymbol{\mathcal{F}}(\boldsymbol{W^c}))\varphi_i^C d\Omega = m(C_i)\partial_t \boldsymbol{W_i^c} + \int_{\partial C_i} (\boldsymbol{\mathcal{F}}(\boldsymbol{W^c}) \cdot \widehat{\boldsymbol{\eta}}))d\sigma \quad (3)$$

with $\widehat{\boldsymbol{\eta}}$ the outward unit normal on ∂C_i, the boundary of the cell C_i.

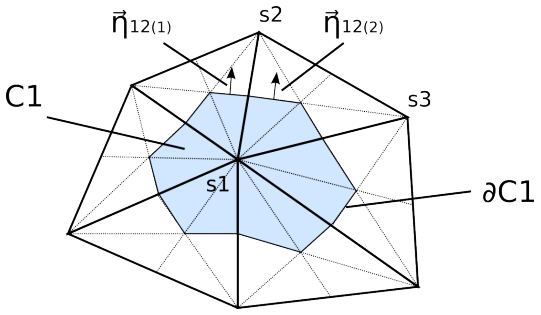

Figure 5: Illustration of a control cell in 2D.

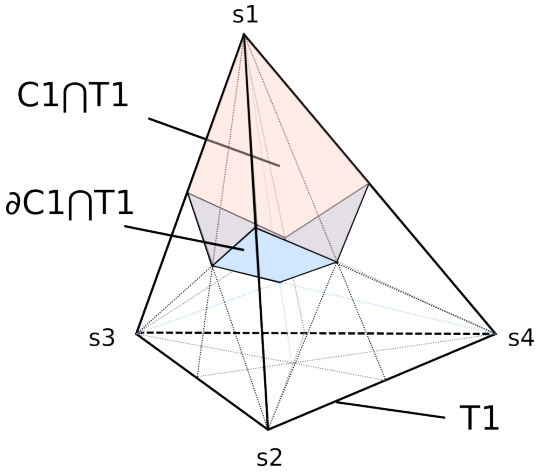

Figure 6: Illustration of a control cell in 3D.

Furthermore, as the domain is discretised,

$$\begin{aligned}
&\int_{\partial C_i} (\boldsymbol{\mathcal{F}}(\boldsymbol{W^c}) \cdot \widehat{\boldsymbol{\eta}}))d\sigma \\
&= \sum_{s_j \in \mathcal{N}(s_i)} \int_{\partial C_i \cap \partial C_j} (\boldsymbol{\mathcal{F}}(\boldsymbol{W^c}) \cdot \widehat{\boldsymbol{\eta}}))d\sigma \\
&\approx \sum_{s_j \in \mathcal{N}(s_i)} \boldsymbol{\mathcal{F}}(\boldsymbol{W^c})_{|ij} \cdot \int_{\partial C_i \cap \partial C_j} \widehat{\boldsymbol{\eta}} d\sigma \\
&= \sum_{s_j \in \mathcal{N}(s_i)} \boldsymbol{\mathcal{F}}(\boldsymbol{W^c})_{|ij} \cdot \boldsymbol{\eta_{ij}}
\end{aligned} \tag{4}$$

The $\boldsymbol{\mathcal{F}}(\boldsymbol{W^c})_{|ij}$ part is modeled by using an approximate Riemann solver for which the associated numerical flux is $\boldsymbol{\Phi_{ij}} = \boldsymbol{\Phi}(\boldsymbol{W_i^c}, \boldsymbol{W_j^c}, \boldsymbol{\eta_{ij}})$.

In the current work, the Rusanov, Steger-Warming [8, 1, 9] numerical fluxes are used.

To reach a high-order approximation in space, a MUSCL reconstruction technique is used. The reconstructed primitive state from $\boldsymbol{W_i^p}$ at the common interface of the cells C_i and C_j is denoted by $\boldsymbol{W_{ij}^p}$: $\boldsymbol{W_{ij}^p} = \boldsymbol{W_i^p} + \frac{1}{2}\alpha_{ij}(\boldsymbol{\nabla W_i^p}) \cdot \boldsymbol{IJ}$. The slope $(\boldsymbol{\nabla W_i^p}) \cdot \boldsymbol{IJ}$ is approximated by $\Delta \boldsymbol{W_i^p} = 2/3\Delta_{|N}\boldsymbol{W_i^p} + 1/3\Delta_{|C}\boldsymbol{W_{ij}^p}$. The nodal slope $\Delta_{|N}\boldsymbol{W_i^p}$ is computed from the nodal P1-Galerkin gradients that is the average gradient of the gradients computed on the elements $T \in \mathcal{T}(s_i)$. The slope $\Delta_{|C}\boldsymbol{W_{ij}^p}$ corresponds to the centered slope $\boldsymbol{W_j^p} - \boldsymbol{W_i^p}$. We denote $\alpha_{ij}(\Delta_{|N}\boldsymbol{W_i^p}, \Delta_{|C}\boldsymbol{W_{ij}^p})$ the limiting coefficient. Note that the reconstruction is performed using primitive variables and not conservative variables.

Finally, the high-order numerical flux becomes: $\boldsymbol{\Phi_{ij}}^{\text{High order}} = \boldsymbol{\Phi}(\boldsymbol{W_{ij}^c}, \boldsymbol{W_{ji}^c}, \boldsymbol{\eta_{ij}})$.

The finite-element method can be retrieved by using the control cell $S_i = T_i$ and by using basis functions on the element as test functions. The polygon T_i is defined as a Lagrange $\mathcal{P}_1$ finite element with a canonical basis denoted by φ_i^T for each vertex s_i. The local basis function φ_i^T is equal to 1 at vertex s_i and zero at the other vertices of the element.

A $\mathcal{P}_1$ approximation of any function f on a polygon T_i can be done by projecting it on the canonical basis φ^T of the polygon:

$$f(\boldsymbol{X}) \approx f_h^{T_i}(\boldsymbol{X}) =< f, \varphi^T >= \sum_{i=1}^{N_s \in T_i} f(i)\varphi_i^T(\boldsymbol{X})$$

A linear approximation on each polygon T_i is selected by using linear polynomial functions defined by the set $\mathcal{P}_1$.

This approximation is used for the density, the velocity and the temperature (or equivalently the internal energy).

By integrating by part, and by using the Green-Ostrogradski theorem, the diffusive term of 2 becomes:

$$\begin{aligned}
\int_{T_i} (\boldsymbol{\nabla} \cdot \boldsymbol{\mathcal{N}}(\boldsymbol{W^c}))\varphi_i^T d\Omega = &- \int_{T_i} \boldsymbol{\mathcal{N}}(\boldsymbol{W^c}) \cdot \boldsymbol{\nabla}\varphi_i^T d\Omega \\
&+ \int_{\partial T_i} (\boldsymbol{\mathcal{N}}(\boldsymbol{W^c}) \cdot \boldsymbol{n})\varphi_i^T d\sigma
\end{aligned} \tag{5}$$

The velocity gradient being assumed to be constant by element, and $\boldsymbol{\nabla}\varphi_i^T$ being a constant:

$$\begin{aligned}
\int_{T_i} \boldsymbol{\mathcal{N}}(\boldsymbol{W^c}) \cdot \boldsymbol{\nabla}\varphi_i^T d\Omega &= \boldsymbol{\mathcal{N}}(T_i) \cdot \boldsymbol{\nabla}\varphi_i^T \int_{T_i} d\Omega \\
&= \boldsymbol{\mathcal{N}}(T_i) \cdot \boldsymbol{\nabla}\varphi_i^T m(T_i)
\end{aligned} \tag{6}$$

By denoting $\boldsymbol{\eta_{T_i}} = -\varphi_i^T m(T_i)$, the viscous numerical flux is defined as:

$$\boldsymbol{\Upsilon_{T_i}} = \boldsymbol{\mathcal{N}}(T_i) \cdot \boldsymbol{\eta_{T_i}} \tag{7}$$

Thus the equation 5 becomes:

$$\int_{T_i} (\boldsymbol{\nabla} \cdot \boldsymbol{\mathcal{N}}(\boldsymbol{W^c}))\varphi_i^T d\Omega = \boldsymbol{\Upsilon_{T_i}} + \int_{\partial T_i} (\boldsymbol{\mathcal{N}}(\boldsymbol{W^c}) \cdot \boldsymbol{n})\varphi_i^T d\sigma \tag{8}$$

The term $\int_{\partial T_i} (\boldsymbol{\mathcal{N}}(\boldsymbol{W^c}) \cdot \boldsymbol{n})\varphi_i^T d\sigma$ concerns the boundary conditions for the viscous terms.

Time integration An implicit second-order time discretization is obtained by using a dual time step approach and a backward time integration. The dual time step approach is similar to the one proposed in [4]. We introduce the computed residual $\boldsymbol{\mathcal{R}_i}$ for the control cell i. By denoting $\delta_1^2\Lambda = \Lambda^2 - \Lambda^1$ the variation of the variable Λ from the state 1 to the state 2, we obtain as second-order implicit time integration scheme:

$$m(C_i)\frac{3\boldsymbol{W_i^{c^{n+1}}} - 4\boldsymbol{W_i^{c^n}} + \boldsymbol{W_i^{c^{n-1}}}}{2\delta_n^{n+1}t} + \boldsymbol{\mathcal{R}_i}(\boldsymbol{W^{c^{n+1}}}) = 0 \tag{9}$$

To solve this problem we introduce a subiteration state of index k, such as:

$$\boldsymbol{W_i^{c^{n+1}}} = \lim_{k\to\infty} \boldsymbol{W_i^{c(k+1)}}$$

The linearization of 9 around the state of index k, with a local time step $\delta_k^{k+1}t_i$ yields:

$$\left(\left(\frac{m(C_i)}{\delta_k^{k+1}t_i} + \frac{3m(C_i)}{2\delta_n^{n+1}t}\right)\boldsymbol{I_n} + \boldsymbol{\mathcal{J}^*}(\boldsymbol{W^{c(k)}})\right)\delta_k^{k+1}\boldsymbol{W_i^c} = -\boldsymbol{\mathcal{R}_i}(\boldsymbol{W^{c(k)}}) + m(C_i)\frac{\delta_{n-1}^n\boldsymbol{W_i^c} - 3\delta_n^k\boldsymbol{W_i^c}}{2\delta_n^{n+1}t} \tag{10}$$

with $\boldsymbol{\mathcal{J}^*}(\boldsymbol{W^{c(k)}})$ an approximate Jacobian of the numerical fluxes. It is composed of an inviscid and a viscous part. The inviscid Jacobian is based on the first-order Rusanov flux. The use of the Rusanov flux, or spectral radius Jacobian approximation, is usually used in matrix-free approaches [5]. The viscous Jacobian is based on the exact Jacobian of the viscous fluxes and is computed as in [2].

The resulting Jacobian matrix is inversed with either a Jacobi or a symmetric Gauss-Seidel iterative algorithm. The use of a delta form for the implicit formulation allows to use an approximated Jacobian without loosing the second order accuracy in space reached with MUSCL extrapolation.

Boundary conditions The boundary conditions are implemented in a weak form through numerical fluxes. The representation of this implementation is given on the Fig. 7.

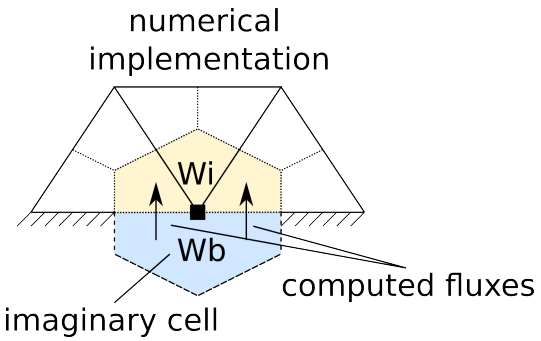

Figure 7: Illustration of the implementation of weak boundary conditions.

The flux computed between the fictive state $\boldsymbol{W_b^c}$ and the boundary (or interior) state $\boldsymbol{W_i^c}$ can be computed with any numerical flux (independently from the interior fluxes). Depending on the boundary condition used, the fictive state $\boldsymbol{W_b^c}$ can be a function of the interior and the exterior state $\boldsymbol{W_o^c}$: $\boldsymbol{W_b^c} = \boldsymbol{W_b^c}(\boldsymbol{W_i^c}, \boldsymbol{W_o^c})$. In this study, the computation of the fictive state is carried out using the Riemann invariants for free-stream conditions, imposed pressure with extrapolation of characteristics for subsonic outlets (characteristic based methods), and symmetry conditions for slipping walls.

The non-slip condition is strongly imposed by setting a zero velocity on the boundary. The Jacobian matrix is thus modified such that the boundary velocity does not evolve.

The viscous boundary fluxes are neglected for free-stream boundary conditions and computed for other cases.

2.2 Possible implementation levels

The numerical methods presented above can be implemented in the platform and plugins with different integration levels. On the one side, we can construct a simple composition based on sophisticated plugins, that basically represent the mesh generator, the solver and the view. On the other side, we can also construct a sophisticated composition based on simple plugins, that represent all single operations performed during the computational scenario. Using the former approach the visual programming tool is not much employed, whereas using the latter approach it is intensively used.

Actually, we choose an intermediate level, that allows to obtain a modular implementation where needed by our research topic. The whole mesh generator is embedded in a single plugin, whereas the solver is partitioned into elementary plugins. Therefore, the loops over finite-volume control cells or finite-elements are defined by the composer, while the local computations of fluxes, Jacobians, time-steps, etc. are embedded into elementary plugins. We obtain finally a quite sophisticated composition, presented in Fig. 8, which contains sub-compositions, such as the one illustrated in Fig. 9 used to compute and store the inviscid flux and the local time step for a given pair of control cells.

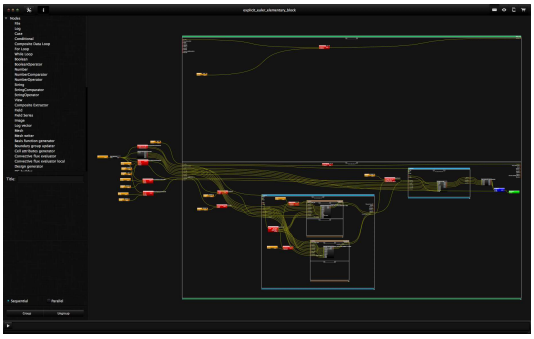

Figure 8: Composition used for flow simulation.

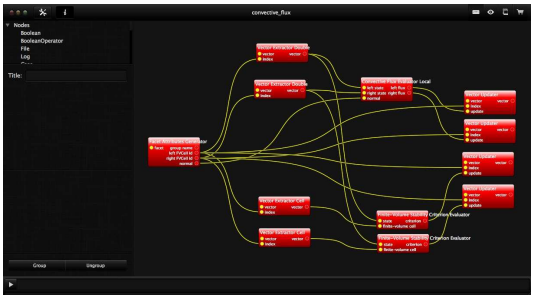

Figure 9: Sub-composition used for flux and local time step computation and storage.

2.3 Interaction parameters

As explained above, the user can interact with the computation by modifying at runtime a plugin or its parameters. In the case of compressible flow simulations, we have so far used three different ways to modify interactively the computations:

- Modification of a plugin to change the algorithm used. For instance, we change the numerical flux or the limiting coefficient function;
- Modification of a parameter of a plugin to change boundary conditions. For instance, we change free-stream velocity, or pressure;
- Modification of a parameter of a plugin to change a numerical method. For instance, we increase or decrease the CFL number used to compute the time step.

All these modifications are performed using a graphical interface in the composer space. Illustrations are provided in the next sections.

3. SOME ILLUSTRATIONS

3.1 Sonic boom crossing for F16 aircraft

As first illustration, we consider the three-dimensional inviscid compressible flow around a F-16 aircraft. The computational scenario chosen here, with which the user interacts, is not the resolution of the state equation, but the construction of a reduced order model from a flow database. A set of 16 computations have been performed and stored, with a grid of about 230,000 nodes, for different values of incidence and free-stream velocity. Then, the visual programming tool is used to construct a scenario that reads these solutions, interpolates linearly the solutions to compute flow variables for a given angle of attack and free-stream velocity, and visualize the data of interest. Using some widgets, the user can modify the angle of attack and the free-stream velocity, while the visualization is updated interactively.

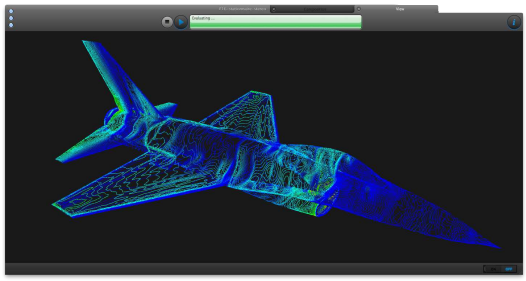

Figure 10: Pressure isolines for F16 aircraft.

To obtain a suitable perception of 3D fields, a volumic rendering feature is implemented in the platform, which lets the user control the color and transparency associated to each field value. The scalar field of interest is evaluated on a cartesian grid from the flow solution and updated as the user modifies the interaction parameters. This task is carried out using GPUs, in order to obtain a satisfactory rendering.

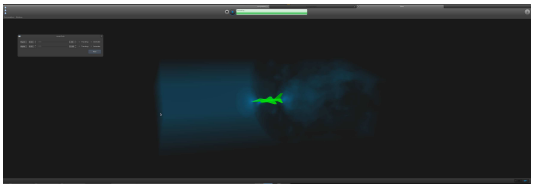

Figure 11: Pressure field in subsonic regime.

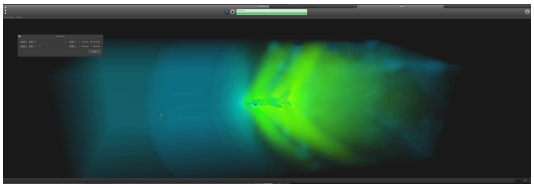

Figure 12: Pressure field in supersonic regime.

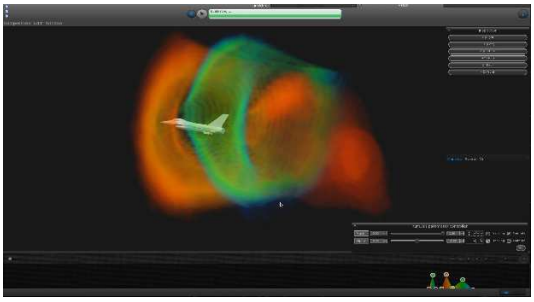

Figure 13: Modification of the volume rendering lookup table to isolate shock waves.

The pressure isolines on the aircraft surface in subsonic regime are illustrated in Fig. 10. The use of the volume rendering technique to obtain a 3D perception of the pressure field is shown on Fig. 11. As the user increases the free-stream velocity, the flow becomes supersonic and three shock waves appear, on the nose, the wing and the tail of the aircraft (see Fig. 12). It is possible to modify the lookup table used for the volumic rendering to isolate the different shock waves, as illustrated in Fig. 13.

3.2 Supersonic ramp

In the previous example, the user interacts with a reduced order model of the flow fields. Now, we would like to illustrate the possible interaction with the PDE solver itself. We consider the inviscid supersonic flow over a $15°$ ramp, for inlet conditions corresponding to a Mach number of value $M_{in} = 2$. The grid counts about 2,500 nodes. An unsteady simulation is performed in order to observe the flow development. During a first phase, one can observe the development of a shock wave and a rarefaction wave (see Fig. 14). Then, the shock is reflecting on the opposite wall and the flow converges to the solution depicted in Fig. 15. At this time, the user modifies the inlet boundary conditions using the widgets located at the left of the visualization space: the Mach number is progressively increased to $M_{in} = 3$. The inlet flow change is progressing in the tunnel and modifies the shock wave characteristics, as illustrated in Fig. 16.

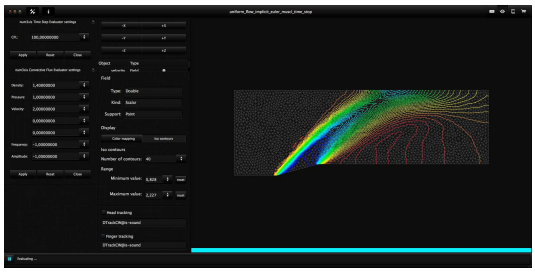

Figure 14: Pressure field: shock wave and rarefaction wave in development.

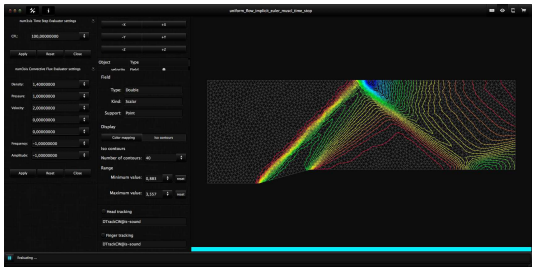

Figure 15: Pressure field: shock wave reflecting.

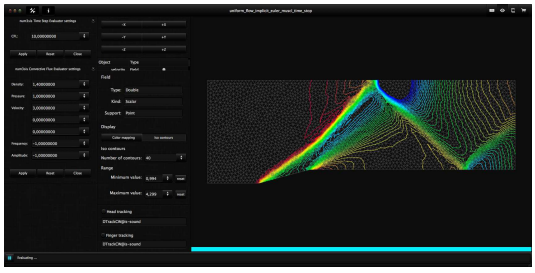

Figure 16: Pressure field: interaction between inlet condition change and shock wave.

This experiment illustrates how the user can interact with a computation at runtime and visualize simultaneously the flow modifications. The change

of boundary condition value has been demonstrated here, but we can consider other types of interaction: the user can for instance modify the boundary condition type or location, or even modify numerical parameters, such as the numerical flux evaluation or the time step.

3.2 Oscillatory jet actuation

Finally, we present the interactive simulation of a viscous compressible flow. We consider the laminar flow over a flat plate, well known as Blasius test-case, and we introduce an oscillatory suction/blowing jet. The free-stream Mach number is $M_{in} = 0.3$ and the Reynolds number is $Re = 100$ (based on the distance δ between the leading edge and the jet location) . The oscillatory jet width is $\delta/10$ and frequency $f = 10$. The grid counts about 8,000 nodes. A second-order time integration is employed, with a physical time step $5\ 10^{-4}$.

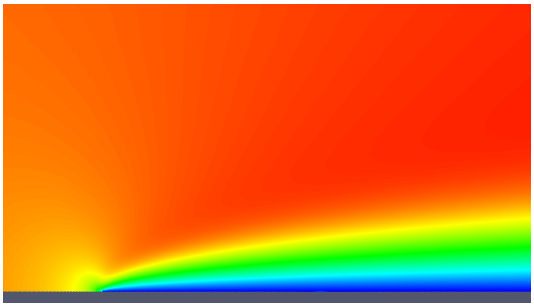

Figure 17: Velocity modulus without actuation (t_0).

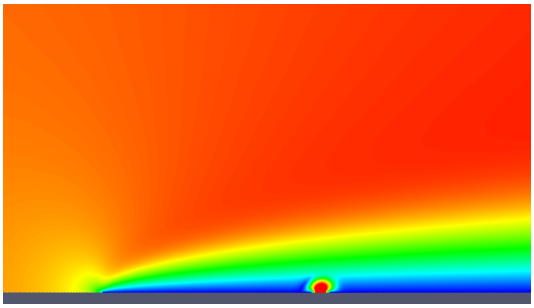

Figure 18: Velocity modulus ($t_0 + 0.015$).

During a first phase of the simulation, no actuation is done. A boundary layer is developing, as illustrated in Fig. 17. Then, at a time t_0, the user decides to activate the actuation. This is done by setting a non-zero actuation amplitude for the plugin that computes the jet boundary conditions (see Fig. 18). For the next time steps, the oscillatory

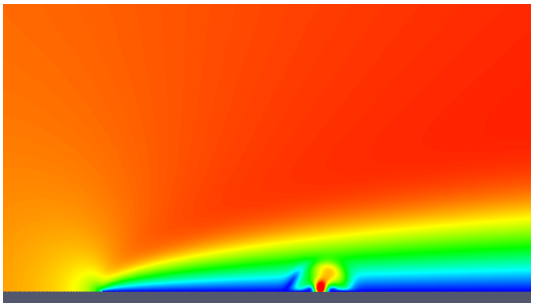

Figure 19: Velocity modulus ($t_0 + 0.040$).

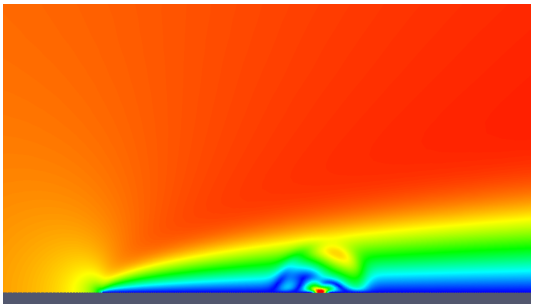

Figure 20: Velocity modulus ($t_0 + 0.060$).

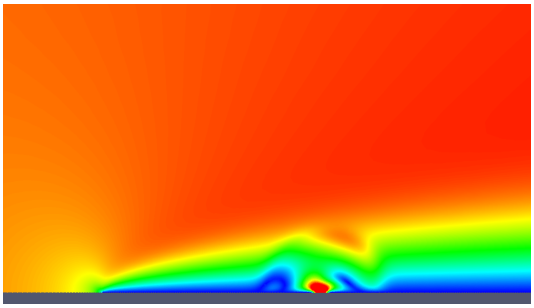

Figure 21: Velocity modulus ($t_0 + 0.085$).

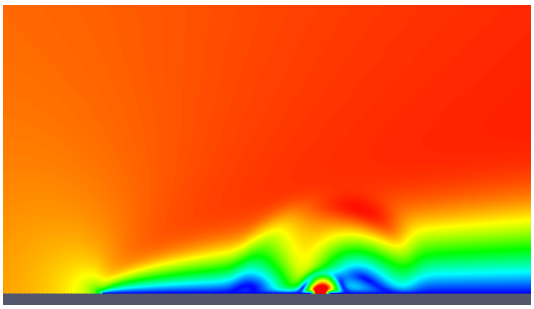

Figure 22: Velocity modulus ($t_0 + 0.120$).

blowing / suction boundary conditions modify significantly the boundary layer, as can be seen in

Fig. 19 to Fig. 22. Although this is not presented here, the user can also modify the actuation frequency interactively.

DISCUSSION AND CONCLUSION

A modern software architecture, based on a platform / plugins system and a visual programming tool, has been used to implement numerical methods for compressible flow simulations. The platform features allow the user to modify interactively the computations by changing some parameters or even some plugins at runtime, while observing the impact on the solution.

Some tests have been carried out, that deal with reduced-order model construction for the 3D inviscid flow around an aircraft, interaction with inlet boundary conditions for the supersonic flow over a ramp, and finally modification of an oscillatory jet for a boundary layer flow.

These tests have shown that interactive computation and visualization in CFD is possible and is beneficial from a scientific point of view:

- a better understanding of physical phenomena or numerical behaviors is obtained;
- the use of CFD tool is more attractive for non-experts;
- the adjustment of physical or numerical parameters is easier.

Of course this approach is limited by the computational facility used. Presently, for a sequential approach, the interactive computation is satisfactory until some dozen of thousand nodes. Therefore, we are presently implementing the use of parallel approaches in the platform, to be able to apply interactive computation to large-scale problems. Our objective is to study interactively turbulent flows, by using simultaneously high-performance computing and virtual reality facilities.

References

[1] BATTEN, P., CLARKE, N., LAMBERT, C., AND CAUSON, D. On the choice of wavespeeds for the hllc riemann solver. *SIAM Journal on Scientific Computing 18* (1997), 1553.

[2] FEZOUI, FATIMA, L., LANTERI, S., LARROUTUROU, B., AND OLIVIER, C. Resolution numerique des equations de Navier-Stokes pour un fluide compressible en maillage triangulaire. Rapport de recherche RR-1033, INRIA, 1989.

[3] FRANCESCATTO, J. *Méthodes multigrilles par agglomération dirrectionnelle pour le calcul d'écoulements turbulents*. PhD thesis, Universit/'e de Nice-Sophia Antipolis, Nice, France, 1998.

[4] JAMESON, A. Time dependent calculations using multigrid, with applications to unsteady flows past airfoils and wings. *AIAA paper 91* (1991), 1596.

[5] KLOCZKO, T. *Développement d'une méthode implicite sans matrice pour la simulation 2D-3D des écoulements compressibles et faiblement compressibles en maillages non-structurés*. PhD thesis, Arts et Métiers ParisTech, 2006.

[6] KREYLOS, O., TESDALL, A. M., HAMANN, B., HUNTER, J. K., AND JOY, K. I. Interactive visualization and steering of cfd simulations. In *8th Eurographics Workshop on Virtual Environments* (2002).

[7] SCHIRSKI, M., GERNDT, A., VAN REIMERSDAHL, T., KUHLEN, T., ADOMEIT, P., LANG, O., PISCHINGER, S., AND BISCHOF, C. Vista flowlib a framework for interactive visualization and exploration of unsteady flows in virtual environments. In *7th International Immersive Projection Technologies Workshop, Zurich, Switzerland, 22–23 May, 2003* (2003).

[8] STEGER, J. L., AND WARMING, R. Flux vector splitting of the inviscid gasdynamic equations with application to finite-difference methods. *Journal of Computational Physics 40*, 2 (1981), 263 – 293.

[9] TORO, E., SPRUCE, M., AND SPEARES, W. Restoration of the contact surface in the hll-riemann solver. *Shock waves 4*, 1 (1994), 25–34.

[10] WENISCH, P., BORRMANN, A., RANK, E., VAN TREECK, C., AND WENISCH, O. Collaborative and interactive cfd simulation using high performance computers. In *18th Symposium AG Simulation (ASIM) and EuroSim. Erlangen, Germany* (2005).